Semantic Web Services and Mobile Agents integration for efficient Mobile Services

Vasileios Baousis
University of Athens, Department of Informatics & Telecommunications, Communication Networks Laboratory, Panepistimioupolis, Ilisia, 157 84 Athens, Greece
Tel: +30 210 7275148
Fax: +30 2107275601
E-mail: bbaous @di.uoa.gr

Vassilis Spiliopoulos
AI Lab, Information and Communication Systems Engineering Department, University of the Aegean, Samos, 83 200, Greece
and
National Centre of Scientific Research "Demokritos", Software and Knowledge Engineering Laboratory (SKEL), Aghia Paraskevi, 153 10, Athens, Greece
Tel: +30 210 6503216
E-mail: vspiliop@aegean.gr

Elias Zavitsanos
AI Lab, Information and Communication Systems Engineering Department, University of the Aegean, Samos, 83 200, Greece
and
National Centre of Scientific Research "Demokritos", Software and Knowledge Engineering Laboratory (SKEL), Aghia Paraskevi, 153 10, Athens, Greece
Tel: +30 210 6503216
E-mail: izavits@iit.demokritos.gr

Stathes Hadjiefthymiades
University of Athens, Department of Informatics & Telecommunications, Communication Networks Laboratory, Panepistimioupolis, Ilisia, 157 84 Athens, Greece
Tel: +30 210 7275148
Fax: +30 2107275601
E-mail: shadj @di.uoa.gr

Lazaros Merakos
University of Athens, Department of Informatics & Telecommunications, Communication Networks Laboratory, Panepistimioupolis, Ilisia, 157 84 Athens, Greece
Tel: +30 210 7275148
Fax: +30 2107275601
E-mail: merakos @di.uoa.gr

Semantic Web Services and Mobile Agents integration for efficient Mobile Services

ABSTRACT

The requirement for ubiquitous service access in wireless environments presents a great challenge in light of well known problems like high error rate and frequent disconnections. In order to satisfy this requirement we propose the integration of two modern service technologies: Web Services and Mobile Agents. This integration allows wireless users to access and invoke semantically enriched Web Services without the need for simultaneous, online presence of the service requestor. Moreover, in order to improve the capabilities of Service registries, we exploit the advantages offered by the Semantic Web framework. Specifically, we use enhanced registries enriched with semantic information that provide semantic matching to service queries and published service descriptions. Finally, we discuss the implementation of the proposed framework and present our performance assessment findings.

KEYWORDS

Mobile agents, Distributed Computing, Ontologies, Semantic Web, Semantic Web Services, Mobile services.

INTRODUCTION

Efficient execution of wireless applications is of paramount importance due to the highly dynamic wireless network conditions. Link outages occur in a near-stochastic pattern, thus, rendering the execution of applications quite tedious and uncertain. Research on mobile computing has longly focused on this specific aspect of wireless application engineering (Pour, 2006). In this paper we adopt the mobile agent paradigm in order to overcome the difficulties discussed above. Surely, this is not the first time that mobile agents are proposed as the vehicle for the implementation of wireless/mobile applications. Their autonomic nature and wide spectrum of characteristics renders the specific technological platform a great enabler for the emerging ubiquitous computing paradigm.

Mobile computing is not the only development that significantly impacts the computer industry nowadays. Service-oriented architectures (SOA) are gradually changing the contemporary structure of the Internet and become a key facilitator for electronic commerce applications and related application domains. We try to incorporate both the discussed technologies in our wireless/mobile computing framework. Mobile agents are dispatched by mobile terminals in order to efficiently and safely satisfy the specific computing needs of their nomadic owner. After securing the autonomicity characteristic in order to progress the required task without the need for the mobile terminal to be constantly online, we try to minimize the service related tasks. Our prime concern lies on the exact identification of the services to be executed at the demand of the user and minimize potential waste of time on unwanted invocations. The accuracy of the service inquiry mechanism has to be improved to really boost the mobile agent and service oriented architecture. To expedite the service querying procedure, and simplify the querying semantics we employ a semantically enriched service registry. A precise definition of the user's requirement is mapped to existing services through a semantically enriched registrar.

In this paper we introduce a novel framework for dynamic discovery and integration of semantically enriched Web Services (WS) with Mobile Agents (MA). The proposed framework is mostly intended for wireless environments where users access Semantic Web Services (SWS) in the fixed network (the terms Web Service (WS) and Semantic Web Service (SWS)

are used interchangeably within this paper). This framework enhances the fixed network with the intelligence needed to dispatch the service requests of the wireless user in an efficient, reliable and transparent manner. The proposed approach enables users to execute multiple services with minimum interaction, without the requirement of being online during their entire session. Additionally, the proposed framework provides better fixed network utilization since unnecessary communication overhead is avoided and reliable delivery of the service results is provided.

The rest of this paper is structured as follows. In section 2 we provide some background knowledge about the implemented technologies, whereas section 3 we discuss relevant prior work. In Section 4, we present an overview of the proposed architecture. Section 5, studies the performance of the proposed framework and presents the results. Finally, Section 6 concludes the paper.

BACKGROUND KNOWLEDGE

In this section we briefly describe the two technologies that are integrated in our proposed framework, namely Web Services and Mobile Agents.

Web Services (WS) provide a loosely coupled infrastructure for service description, discovery and execution. In the traditional WS model, service requestors find the appropriate service by placing a request to the service registry, often implemented with UDDI (Universal Description, Discovery and Integration), obtain the result(s) - public interfaces of the chosen service(s) (expressed in WSDL - Web Services Description Language) and, finally, send SOAP (Simple Object Access Protocol) messages to WS provider(s).

The main problems experienced in these interactions are:

- UDDI guarantees syntactic interoperability, and does not provide a semantic description of its content. UDDI is characterised for its lack of semantic description mechanisms, such as semantic interoperability, explicit semantic models to understand the queries and inference capabilities. UDDI service discovery is performed primarily by service name (keyword matching), but not by service attributes/capabilities. UDDI tModels may be regarded as a vocabulary where service descriptions are unstructured and intended for human comprehension. Different services with the same capabilities can thus be categorized in different business categories.
- WSDL is XML-based and used to specify the interface of a WS. It describes the information being exchanged (structure of the SOAP messages), how this information is being exchanged via interactions with the WS (transport protocols) and where the WS is located. However, WSDL does not contain any information about the capabilities of the described service and as such service discovery based on service capabilities or semantics cannot be performed.

Several efforts have been made to address the lack of expressiveness in WSDL in terms of semantic description that fall into the area of the Semantic Web (SW). SW is a vision in which web pages are augmented with semantic information and data expressed in an unambiguous manner and can be understood and interpreted by machine applications and humans alike (Berners-Lee, 2001). This requires means to represent the semantics of the exchanged data so that it could be automatically processed. This requirement is met with the use of ontologies. Ontologies facilitate knowledge sharing among heterogeneous systems, through explicit formal specifications of the terms used in a knowledge domain and relations among them (Gruber, 1993). Ontologies are machine-understandable and, as such, a computer can process data, annotated with references to ontologies. Through the knowledge encapsulated in the ontology, a computer can deduce facts from the originally provided data. The use of ontologies enables systems to share common understanding of the structure of information and

reuse of domain knowledge, make domain assumptions explicit and separate domain knowledge from the operational knowledge.

Currently, several upper ontologies (terminology in the form of an ontology) have been proposed for Web Service description. The first was DAML-S (McIlraith, 2003), which was based on DAML+OIL ontology language. When DAML+OIL evolved to the widely accepted OWL (Web Ontology Language) family of languages, DAML-S was replaced by OWL-S (OWL-S, 2007). Still, OWL-S does not constitute a commonly accepted description language, there are also other languages proposed such as WSDL-S (Verma, 2006), WSMO (Roman, 2005) and SWSO (SWSL Committee, 2007). All these languages differ in terms of expressiveness, complexity and tool support.

OWL-S, which is adopted in our work, has well defined specifications by the W3C (World Wide Web) consortium(OWL-S, 2007) and is widely accepted by the scientific community. OWL-S ontology implicitly defines message types (as input/output types of processes) in terms of OWL classes, which allows for a rich, class-hierarchical semantic foundation. Specifically, OWL-S models the web services via a three-part ontology: (i) a service profile describes what the service requires from users and what it gives them, (ii) a service model specifies how the service works, and (iii) a service grounding provides information on how to use the service.

With OWL-S, SWS are described in an unambiguous manner allowing for a potential service requestor to place a capability search in a service registry rather than a keyword search in UDDI registries. Registries that offer such capability search functionalities are called Semantic Web Registries (SWR).

The most representative matching techniques used are detailed in (Tsetsos, 2007) and are summarised below:

- *Semantic Capability Matching*. The basic idea is that an advertised service matches a requested one, when all the inputs (respectively outputs) of a requested service are matched by the inputs (respectively outputs) of the advertised service. For this purpose Description Logics (DL) reasoning services are exploited for inferring relations between ontology concepts.
- *Multi-Level Mapping*. Matching is performed in many levels, not only between input and output descriptions. Service categories or other custom service parameters (e.g., OoS) may be exploited. The result is a more efficient ranking of the matched services.
- *DL Matching with Service Profile Ontologies*. Each service and query are represented as ontologies following the DL formalization. Hence, a DL reasoner is utilized for placing the query concept in its proper position in each service ontology description (e.g. as a subconcept). Then specific rules are applied for computing the degree of relevance between the query and each service description.
- *Information Retrieval Based*. In this category, vector space techniques (Raghavan 1986) 0are utilized for locating the most related service to a provided query.
- *Graph-based Approaches*. Ontologies representing services are transformed into directed graphs and various algorithms accomplish the matching between such graphs.

There is a plethora of tools that provide Semantic Web Services functionalities (e.g., OWLS-MX (OWLS-MX, 2007)**Error! Reference source not found.** and TUB OWLSM (OWLSM, 2007), each one implementing a portion of the above matching techniques. In our work, we adopted the OWL-S/UDDI Matchmaker tool (OWL-S/UDDI Matchmaker Web Interface, 2007)0, (Paolucci, 2002), (Srinivasan, 2004), which mainly implements techniques from the first aforementioned matching technique.

Mobile agent technology is one of the most promising technologies for communicating and managing functional components comprising a mobile service (Lange, 1998) (Wooldridge, 2002). A MA has the unique ability to autonomously transport itself from one system to another. The ability to travel allows a MA to move to a system that contains an entity (-ies) with which the agent wishes to interact and take advantage of being in the same host or network with the collaborating entity. MAs can operate synchronously and asynchronously, and are equipped with the appropriate intelligence and knowledge to dynamically accomplish their task without user interaction. MAs are not trying to replace traditional ways of communication but to enhance the functionality and operation of the involved service entities. Researchers agree that MAs are not always the best solution and a combination of the MA, client-server and remote execution paradigms delivers the best performance with respect to network operation metrics like bandwidth, response time, and scalability.

RELATED WORK

In this section, we provide an overview of the related work performed in the areas of semantic WS and multi-agent systems and, especially, on research activities that integrate these two technologies.

In (Ishikawa, 2004) and (Ishikawa, 2004b), BPEL (Business Process Execution Language) is used to form simple rules to describe MA physical behaviours (e.g., migration and cloning). Such simple rules are separated from the integration logic, allowing for addition or change of physical behaviours without modification of the BPEL description. This separation is considered helpful in dealing with the dynamic environment of WS, however, the discussed framework supports actions only in case of predefined events. The implemented rules do not consider dynamic events that might be generated during WS invocation and MA roaming. Moreover, and importantly, directory services and multicast protocols are assumed pre-existing and not discussed. The discussed framework refers only to interactions occurring among MA and WS without considering the interactions of the MA and service registries that have equal importance in such a system. Finally, the system description does not include any implementation, hence benchmarking is not considered.

There are several proposed models that adopt BPEL4WS (Business Process Execution Language for Web Services) as a specification language for expressing the social behaviour of multiagent systems and adapt to changing environment conditions (Bulher 2003), (Bulher 2004). Moreover, in (Montanari, 2003) and (Montanari, 2003b), the authors propose a policy based framework for flexible management and dynamic configurability of agent mobility behaviour in order to reduce code mobility concerns and support rapid mobile code-based service provisioning. Policies specify when, where, how and the parts of the agent that will perform a given task (e.g., migrating to a host and invoke a service). However, these models do not provide for the semantic description of the WS involved in their systems. Other proposed frameworks adopt DAML-S for describing the WS, thus, allowing for service capability search and matching (Gibbins, 2004), (Kagal, 2002). However, the proposed systems are intended for fixed networks, and problems related to the wireless environments are not considered.

An agent based approach for composite mobile WS is proposed in (Zahreddine, 2005), where three methods for compositions are discussed: parallel, sequential and a hybrid of these two. The service composition scenario is that a user with a wireless device places a request to execute a WS and a MA executes the service on behalf of the user by moving to the service registry, query the registry, get service description (in WSDL), and finally invoke the service. Service execution, depending on the WS itself, is performed with one of the aforementioned composition methods. This approach does not consider semantic information describing the involved WS, thus services are selected by simple keyword queries to the UDDI registry. Additionally, it does not include mechanisms to decide which composition approach to follow.

Integration depends upon the nature of the WS (if the service is a composition of other services it must be accessed sequentially, if not, then in parallel). A similar approach is proposed in (Cheng, 2002), with the difference that a personal and a service agent are used to perform the task of the MA described in the previously mentioned approach.

Moreover, in the research literature it has proposed the use of Asynchronous Web Services (AWS) in order to access WS with asynchronous interaction. AWS can be used where the standard Web Service Business Model has some limitations, as described in (Brambilla, 2004): a. when service time is expected to be too long b. when response time is not predictable and c. when users may not be continuously online. The most common way to achieve asynchronous calls to Web Services is by using a correlation or conversation ID ((Brambilla, 2004) and (Huang, 2003). This unique ID is assigned initially by the Web Service providers to each web service transaction and it is passed in each exchanged message between the conversational parties. This way, the client is able to perform correlation and to retrieve application data related to current conversation. The drawback though of such an approach is the production of possible mismatches. Specifically, if multiple asynchronous Web Service calls happen in the context of a single conversation, responses might not be able to be unambiguously related to their requests(Brambilla, 2004).

FRAMEWORK ARCHITECTURE

The proposed framework consists of the mobile user that uses SWS, the MA representing the user in the fixed network, the service registry and the SWS provider. The last two entities are implemented as stationary agents. According to the service implementation scenario (Figure 1), a mobile user accesses the proposed system and places service requests specifying some criteria. Subsequently, the system creates a MA (step 1) that migrates to the registry to find the WS that best meets the user requirements (step 2). Service registry allows for a capability search to be performed, since it is enriched with semantic information. The MA, after acquiring the WS listing and technical details, migrates to service provider(s), invokes the WS, collects the results (steps 3-5) and returns to the service requestor to deliver the results to the mobile user (step 6). In the presented scenario the SWS that matched the service request were three thus MA migrates and invokes these three services (steps 3-5). If the service request matched more than three services during the step 2, the MA would migrate to all these matched WS (Figure 1 would include more steps). The advantages of this scenario is that the MA has the necessary intelligence to invoke only the best matched service(s) and unnecessary service invocations are avoided leading to better network utilization, and the wireless user is not required to be online and may obtain the results on future time.

In the proposed framework the route of the agent may vary, depending on the service requestor preferences and the network topology. As explained below, the user may dynamically force his MA to send its clones to the providers, invoking the services in parallel, rather than serially migrate to each one. Moreover the user may force the MA to implement different service execution strategies (e.g., execute all services locally or remotely, change timeout limit), during its itinerary and execution of service(s).

Our framework consists of the following functional components: (1) User Service Requestor (USR) who is the user that invokes a SWS, and the Client System, the system in the fixed network that provides user access to the SWS, (2) Mobile Agent which is the representative of the user in the fixed network (3) Provider Stationary Agent (PSA) which is a stationary agent that resides in the host offering a certain WS (its implementation is optional), (4) Registry Stationary Agent (RSA) which is a stationary agent that acts as a broker between the MA and the service registry (its implementation is optional), (5) Semantic Web Services Registry (SWSR), the registry where the service providers advertise their services (6) Web Service Provider (WSP) which provides the WS to interested users. Their structure and functionalities

are described below. In the end of this section we provide a service implementation scenario, presenting all possible supported service invocation alternatives.

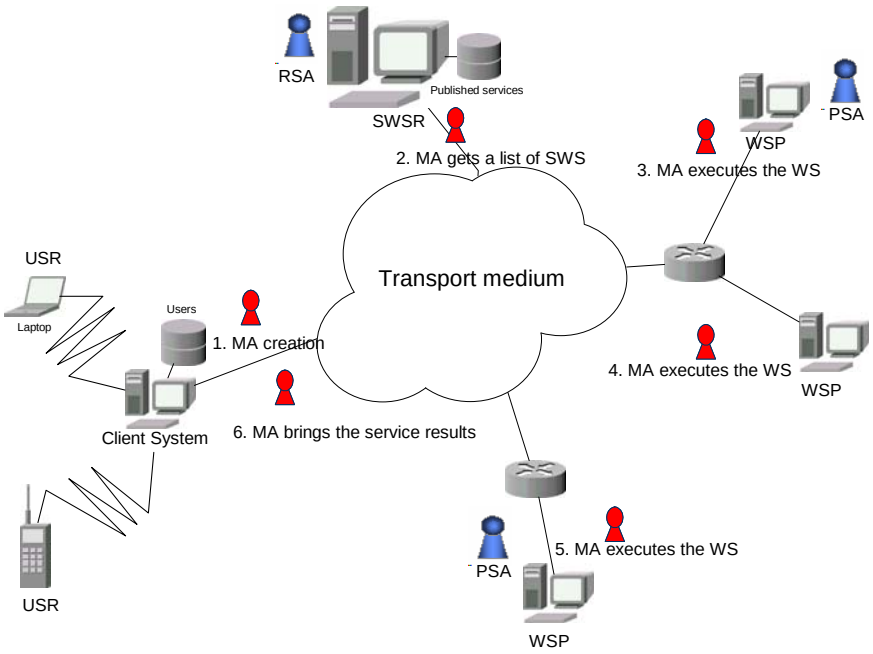

Figure 1: Service implementation scenario

User Service Requestor (USR)

USR is the client that invokes a WS. USR logs into the Client System, which communicates with the agent platform using IIOP (Internet Inter-ORB Protocol). The agent platform is responsible for creating and handling MA, according to user specifications. The Client System is implemented in JSP/Servlet technology, and many users can be accommodated without having java runtime environment (JRE) or the MA Platform (MAP) installed on their device. The only requirement is a browser to access the Client System.

The Client System offers services to clients like: account creation, user login/logout, service invocation policies profile editing, and control of existing agents. Moreover, the administrator is allowed to add/remove/edit user properties/profiles. Finally, users' service invocation policy profiles are serialised and stored into the server's database that enables the seamless and transparent provision of services.

The proposed framework is in addition able to communicate with mobile devices that are capable of hosting JADE/LEAP (Lightweight Extensible Agent Platform) (JADE, 2007). LEAP is an extension of JADE that enables MAs to be executed on wireless devices with limited processing capabilities. In such a case the MA is spawned on the mobile device, gathers the user preferences/specifications either from this device or from the Client System. The behaviour of the system and of the device created the MA is exactly the same.

Mobile Agent (MA)

The MA is the representative of the user in the fixed network and is capable of roaming, finding and executing services and delivering results to the user. The MA may also spawn clones that execute the selected WS in parallel to minimize the total processing time. Clones can migrate and invoke simultaneously the chosen WS and return to the service requestor with the

results. The MA has the following components: (1) data state, (2) code, (3) migration and cloning policies, (4) matching engine, and, (5) policy management component (Figure 2).

The proposed MA architecture is based on that the logic of the MA has to be separated from its implementation, enabling the modelling of the MA to be platform independent. That is to say, the physical behaviours of the MA are portable to any MAP (Jade, Grasshopper, etc.). Below, we describe the components of a MA.

The data component contains the information collected by the MA from the SWS invocations. Several compression algorithms may be applied in order to reduce the size of the collected information. The migration and cloning policies component specifies the autonomous behaviour of the MA. It should be noted that the social behaviour of the MA (migration, cloning) is separated from integration logic and code implementation. This separation is accomplished with user's specified invocation policies that govern the behaviour of the MA, being external and independent of its code and integration with the WS. Moreover, the matching engine component is responsible for post-processing the service registry query results, i.e. confirm the availability of the service providers prior to agent migration.

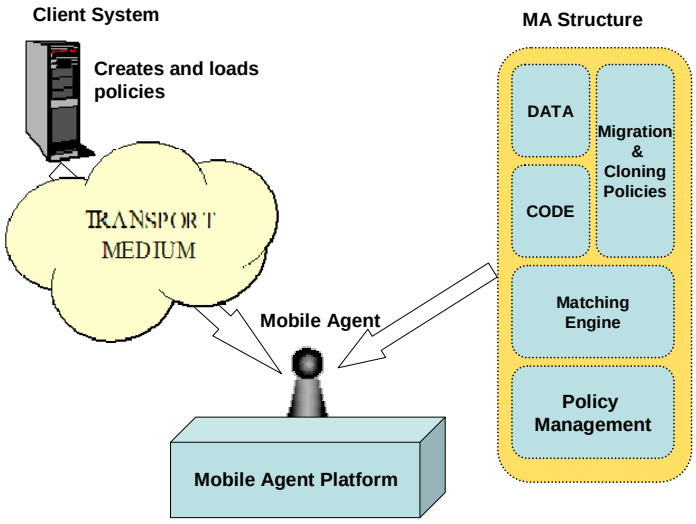

Figure 2: Mobile Agent Structure

The policy management component is responsible for the MA external communication and the transparent installation of policies into the agent's repository.

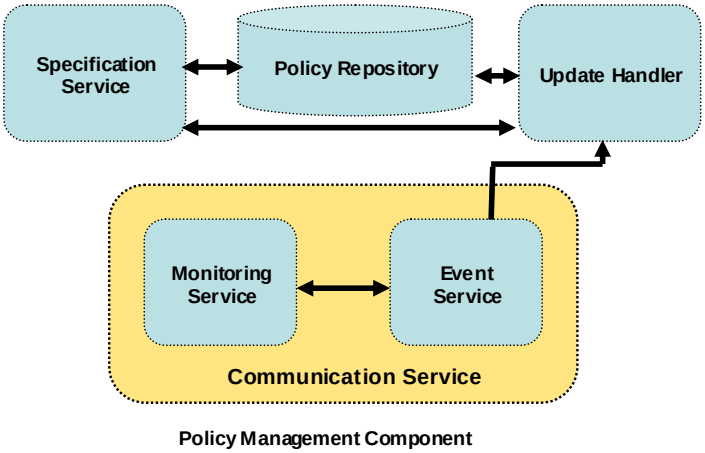

Figure 3: MA Policy Management Component

As shown in Figure 3, the policy management component provides four services, namely communication, update handler, specification and policy repository. Policy repository contains the user preferences and policies that govern the behaviour of the MA. Communication service enables the MA to interact with the client and other network entities. Such functionality is achieved through the monitoring service which filters the messages coming from the client system and through the event service which handles events concerning policy changes. When a policy change occurs, the update handler is notified to update the policy repository. Specification service is responsible for fulfilling this task.

The agent's policies determine its physical behaviour while roaming in the network and executing WS. Currently, the MA considers the policies (Table 1), which are Boolean and numerical variables. Agent policies are expressed in XML and stored in a serialised format into the Client System database. For each registered user there is an associated policies file, to provide personalized WS access.

Table 1: Policy names and their respective meaning

Policy name	Type	Description
<Migrating> and <Cloning>	Boolean	MA's ability to migrate to another host and spawn clones respectively.
<retryTimes>	Numerical	The number of attempts that MA will perform when a WS is unavailable.
<timeBetweenReattempts>	Numerical	The time that MA will wait between consecutive reattempts.
<suspendWhenFinished>	Boolean	States if the user wishes (dis)-connected operation and what the MA should do when returns to the Client System (suspend its state and wait user to connect back or to deliver immediately the service results).
<rollBackBehaviour>	Boolean	Specifies in a case of failure if a roll back solution will be followed.

<maxNumberOf-Hits>	Numeri-cal	The maximum number of services to be invoked.
<minNumberOf-Results>	Numeri-cal	The minimum number of results when searching the semantically enriched service registry (it is accomplished through the similarity level that is returned from the semantic engine that enables the system to always return a result, even though it does not always satisfy completely the request).
<pingServer>	Boolean	States that the MA should check if the targeted service provider is alive, before MA starts the mi-grating process to this host.
<migrate-ToServer>	Boolean	Specifies if the service will be invoked locally or remotely.
<remoteCall>	Boolean	MA invokes the chosen services using SOAP/RPC (remotely from other host) without migrating to each provider.
<callThroughSta-tionary>	Boolean	Indicates if communication between WS and MA will take place with or without the Provider's Sta-tionary Agent (PSA).
<HitAllServices>	Boolean	Forces the MA to invoke all retrieved services from service registry.
<cloneToServer>	Boolean	Enables the agent to decide whether to serially migrate to each located service provider or sent clones to accomplish the task in parallel and return service results to their parent agent and then are self destroyed.
<UserDisconnect-edOperation>	Boolean	States that user wishes to retrieve results in a future time, by reconnecting to the Client System.
<timeBetweenRe-attempts>	Numeri-cal	The time that MA will wait between consecutive reattempts.

Provider Stationary Agent (PSA)

PSA is a stationary agent that resides in the host offering a certain WS. Its purpose is to wrap the functionality of the WS. The PSA is created and maintained by the service provider. PSA communicates with the service providers through protocols specified for WS invocation and interaction (e.g., SOAP). When the MA migrates to a host offering a WS with a PSA, it obtains the results through the PSA. This communication is performed with Agent to Agent protocols (either Remote Method Invocation or exchanging FIPA/ACL messages using a FIPA/Message Transport Protocol-MTP (FIPA, 2007), instead of the resource consuming SOAP. In this approach, the MA need not be SOAP fluent, thus leading to a lightweight implementation. It should be noted that this implementation maintains the platform independence as far as it concerns the SWS provider. This is due to the fact that the PSA wraps the SWS functionality, and is also SOAP fluent and exposes the same SWS's functionality in a native form to the MA.

Figure 4 presents the structure of a PSA. PSA interface exposes the available methods of the SWS as they are described in OWL-S. PSA consists of two parts: (1) its data state, and, (2) its code. PSA methods are multi-threaded to accommodate and simultaneously serve multiple MAs.

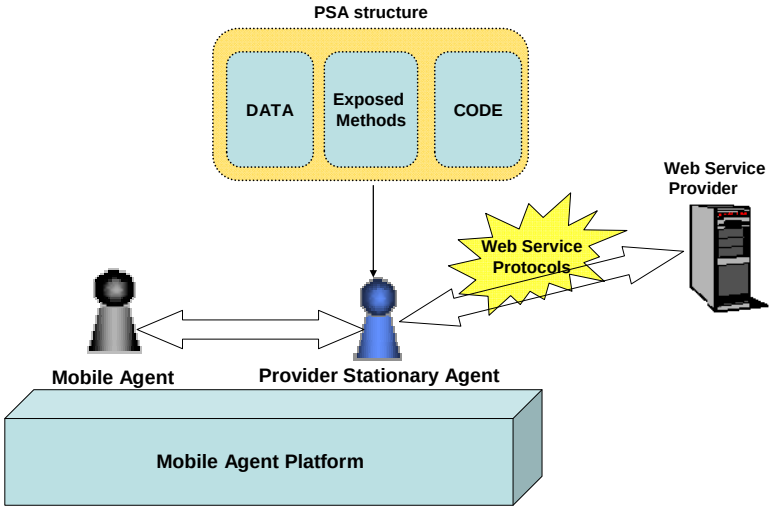

Figure 4: Provider Stationary Agent Logic

Registry Stationary Agent (RSA)

RSA is a stationary agent that acts as a broker between the MA and the service registry (Figure 5). RSA implements part of the registry's functionality and serves MA's requests. By using a RSA in the WS registry, MA does not have to be aware of the implementation specific functionalities of the registry. Thus different service registries can be used as long as RSA acts between WS registry and MA. The proposed framework can be used with different registries that are currently available (e.g., ebXML (ebXML, 2007), OWLS-MX (OWLS-MX, 2007), TUB OWLSM (OWLSM, 2007).

Semantic Web Services Registry (SWSR)

The SWSR (Figure 5) consists of the RSA, the matchmaking tool and the UDDI registry. The matchmaker (OWL-S/UDDI Matchmaker Web Interface, 2007) is a tool which enhances the UDDI server by adding capability-based discovery. In combination with Racer (RACER, 2007), it processes the ontologies expressed in OWL. Service advertisements are first processed by the UDDI server and, if any semantic information is contained by them, they are passed to the OWL-S matchmaking engine. Finally, the engine processes service queries and returns the results to the UDDI server, which in turn, communicated with the requesting service client.

The matching algorithm used by Matchmaker to match a service request to a service advertisement is based on matching all the outputs of the first to the outputs of the latter, and all the inputs of the latter to the inputs of the first. The matching degree (between I/O of a request and I/O of an advertisement) depends on the correlation of the domain ontology concepts associated with these I/O. Matchmaker specifies four matching degrees (in decreasing order of matching importance): Exact, plugin, subsumes, fail. The query language used in the registry is the standard query language of Racer that has its basis on LISP. It is powerful and has more functionalities than standard OWL query languages.

Matchmaker is a tool that integrates seamlessly with registries such as UDDI. In our system we used a local implementation of UDDI, called jUDDI (jUDDI, 2007). JUDDI is a web ap-

plication for Apache Tomcat. The matchmaker tool is responsible for the mapping of the OWL-S service description to JUDDI. Matchmaker is plugged in JUDDI and is available in two versions, a Web-based and a standalone version. The standalone version provides a matching engine and a client API for invoking this engine. In our framework we used the standalone version of Matchmaker. An extensive description of matchmaker can be found in (Paolucci, 2002) and (Sycara, 2004).

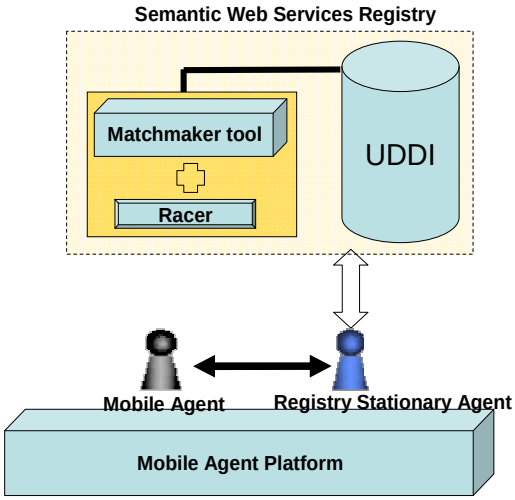

Figure 5: Semantic WS Registry

Web Service Provider (WSP)

The WSP provides the WS to interested clients. It maintains a description of the WS expressed in WSDL and OWL-S. Figure 6 depicts the WSP and their supported functionalities. Service invocation by the MA depends on the OWL-S description of the service. In our framework, service invocation by MA is performed either directly or through the PSA. In the direct access case, the agent has to be SOAP fluent, a fact that increases the size of the MA when moving over the network. Inside the OWL-S description of the WS, it is indicated if a PSA wraps the functionality of the service to allow the roaming MA to interact with the PSA instead of the SWS.

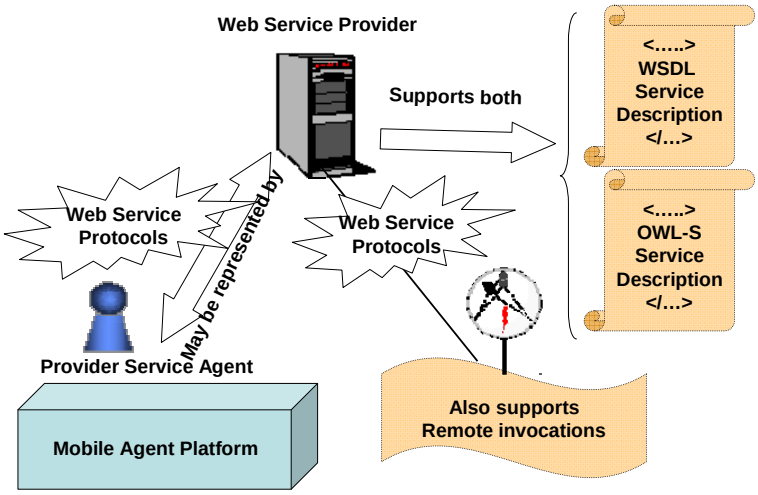

Figure 6: Web Service Provider

As mentioned above, OWL-S is used to enhance the expressiveness of WSDL in terms of semantic information. For this reason, in our framework, WS are described both in WSDL and OWL-S. WSDL is used to describe the technical details (information included in the service grounding) and OWL-S is used to specify the input and output ontologies, thus, enabling an advanced service capability search (service profile and model). Upon retrieval of the desired services from the registry, the WSDL description is used to find the necessary definitions for its successful invocation.

As already mentioned, the SWS provider can expose a PSA to act as his delegate and interact with the user's MA. This is revealed to the MA through the OWL-S description. If this is not the case, the MA infers that no PSA is offered and the service should be accessed directly.

Service Usage Description

In this section a functional description of the proposed framework is provided, through a service scenario. According to this scenario a USR needs to find and invoke a certain WS by using a mobile device. Therefore, he/she connects to the Client System, the platform front-end. After a successful registration, the USR sets the desired criteria for the WS. The user also defines the MA service invocation policies and forces the MA to follow a certain policy while roaming throughout the network. Subsequently, a MA is created, equipped with the user's unique ID, service invocation and agent behavioural policies, to represent the user in the fixed network and dispatch his service requests. The aforementioned policies are passed to the MA in XML format and stored into its policy repository. Remind that the update handler has the authority to change these policies, according to the messages that the event service may receive from the USR or other network entities. The MA, after creation, migrates to the SWSR. The SWSR provides SWS descriptions and allows service capability search. When the MA arrives at the service registry, it communicates with the RSA, which queries the registry on behalf of the MA. RSA finds the service(s) that meet the user needs and delivers them to the MA, which decides on the next step according to its specified service invocation and agent behavioural policies.

The MA may follow several WS invocation alternatives and these are listed below:

1. Poll the servers where the services are located to check their availability, in order to migrate only to those that are alive. In this way, the MA is released from the burden of migrating to a malfunctioning remote server .This strategy improves the overall performance of the framework by avoiding unnecessary migrations.
2. Try to invoke the services from remote and not migrate to the provider. Remote invocation or migration of MA is specified in the MA policies. Specifically, depending on the size of the MA or the distance between its current location and the location of the provider, it might be preferable not to migrate, but remotely invoke the WS.
3. Migrate to the WSP and collaborate with the PSA. The MA invokes the service and obtains the results through the PSA.
4. Migrate to the WSP and directly invoke the WS. This option requires the MA to carry additional code libraries. The implementation of the WSP is much simpler and straightforward since there is no change in the traditional WS implementation model.
5. Finally, to send clones to each WSP, instead of migrating serially to each one. This scenario results to a parallel invocation of WSs where each MA clone invokes one WS. In this way, the overall service invocation time is reduced in comparison to the previous service invocation alternatives.

All these service invocation alternatives are decided at runtime through the user's specified service invocation and agent behavioural policies. When the MA(s) have collected the results, there are two options depending on the selected policies:

1. When the MA invokes all the services, it migrates back to the Client System. If the user is logged in the system, the MA passes the results to the user. Otherwise, the MA waits for the user to login and ask for the service results.
2. When the MA clones have been used for service invocation, they return to the Client System and deliver service results to the father MA. After this interaction the MA clones are destroyed. Consequently, the father MA delivers the services results to the user in a similar way to the previous case.

When the USR obtains the results, he may ask the MA to repeat one of the above scenarios by changing, if necessary, its policies, or he may cancel the execution of the agent. The USR may also, at any time, search for the agent, instruct him to return or cancel its execution at runtime.

A practical example of the proposed framework usage could be to book a trip from a place A to a place B and probably specifying some preferences on each action (e.g the flight to have an intermediate stop to location C). The user requests this service by specifying his preferences and a MA fulfils this request. The MA has the intelligence to query the Semantic Web Service Registry and with the help of the semantic matchmaking capability of the registry to retrieve the most accurate service (that meets the requirements of the user), to invoke this service and provide synchronously or asynchronously the results to the user without his/her online presence. The semantic expression of the WS to the registry and the unambiguous matching of the user's criteria with the capabilities of the available SWS, leads to as much accurate results as possible, as well as maximization of the recall.

Maximization of the matching performance is of paramount importance, since it might be possible that the exact service does not exist in the registry catalogue, but still the most accurate result has to be retrieved. The matching algorithm that is used in the registry ensures this fact: it defines a flexible matching mechanism based on the OWL's subsumption mechanism. The degree of match between the request and the available services depends on the match between the concepts of the two ontologies. Specifically, the matching mechanism relies on a semantic matching between concepts, rather than a syntactic one. Let us consider the practical example mentioned above. The user wants to book a "flight" from location A to location B, and the registry contains a service that does not match exactly with the user's request in the sense that in the service advertisement the output is specified as "trip", as shown in Figure 7. Although there is no exact match between the output of the request and the advertisement, the

matching algorithm recognizes a match, since “trip” subsumes “flight”. This is a clear advantage over a simple string matching-based UDDI registry.

Figure 8 illustrates the semantic matching process that ensures efficient service retrieval since compares concepts that are unambiguous specified on three levels, service profile, model, and grounding.

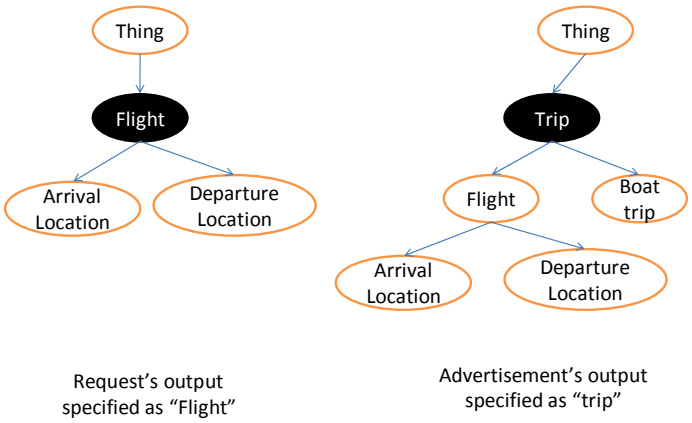

Figure 7: Semantic Matching

If traditional methods of WS invocation are followed, this booking would be performed as follows: the user would browse to a UDDI registry, request all the WSs that provide a flight booking service and get the results. Due to the lack of semantics in the service registry (UDDI does not supports for WS semantic annotation) and to keyword search that is performed in such registries, the user would obtain WSs that provide booking services, probably either irrelevant WSs or services that are not classified according to the relevance of the query. As a result, the user would need to sequentially or randomly invoke each service till he finds a service that best meets his requirements. This interaction requires the on-line presence of the user during the whole interaction.

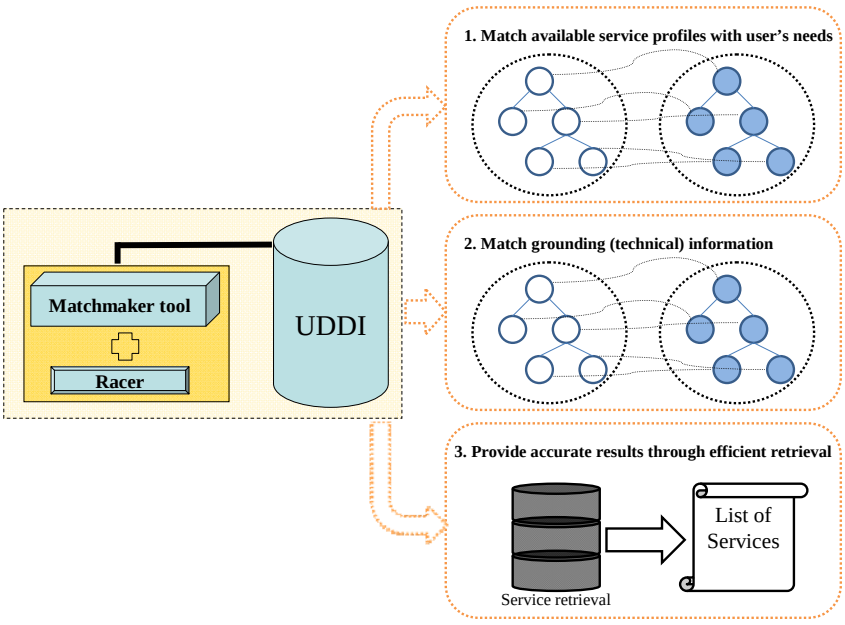

Figure 8 : Web Service Retrieval

PERFORMANCE EVALUATION

In this section we discuss the performance evaluation and present the results of the proposed system. Specifically, we compare the performance of our framework against the traditional business model of WS provision. In the following description, the term "conventional WS Business Model" (WSBM), refers to the model where a user requests a service to be executed and the system dispatches (either automatically or with user intervention) the request by discovering the appropriate service(s) from the service registry, and then, sequentially, invokes these WS, receives and forwards/presents to the user the service results. All communication among the involved network entities is performed with SOAP. Moreover, in our framework the mobile agents are implemented on JADE (JADE, 2007) MA platform. We have developed and tested the following system:

a. A WS system implemented with the "Conventional WS Business Model" (**WSBM**).
b. Our framework (Semantic Web Services and Mobile Agents) (**SWS& MA**)

The SWS logic implemented in our experiments is as follows: the SWS have an extensive service description, stating unambiguously their capabilities in OWL-S. This description is published in the registry (SWSR). However, the SWS internal functionality is fairly simple, returning a pre-specified data volume subject to the service request. In our trials, these service results are 1 KB, 10 KB, 100KB and 1 MB. Moreover, six SWS have been implemented and distributed in the testing network.

In the performance evaluation scenario, a user requests a service, specifies his/her preferences and each of the above systems dispatches this request to the service registry. The service registry in the WSBM is a simple local UDDI providing a keyword service search on each service request, whereas in the SWS&MA system the registry is (as described in paragraph 0) offering a service capability search to the placed service requests. In our evaluation the description of SWS had small differences in the OWL-S descriptions. As a result, in the WSBM system, the service search to the UDDI registry had an average of three matches per service search/request. Contrary to WSBM system, in the SWS&MA system the MA had the necessary intelligence and knowledge to filter the results from the semantic registry and invoke only a SWS where its semantic description matched the service request and user's preferences. Consequently, in the WSBM system we considered the averaged time this system requires to execute a service and we multiplied that by 3 (the average service results from the registry), whereas in SWS&MA system we consider the average time that is needed to invoke only a SWS. Moreover, in the SWS&MA system, the average time need was used from all the system variations to execute a SWS. These system variations are: (a) a system that uses MA cloning, (b) a system that uses PSA, and (c) a system that uses both MA cloning and PSA.

The testing platform we used is depicted in Figure 9. The system is a LAN that is composed of two workstations and a portable PC, all connected to the Internet through University's MAN.

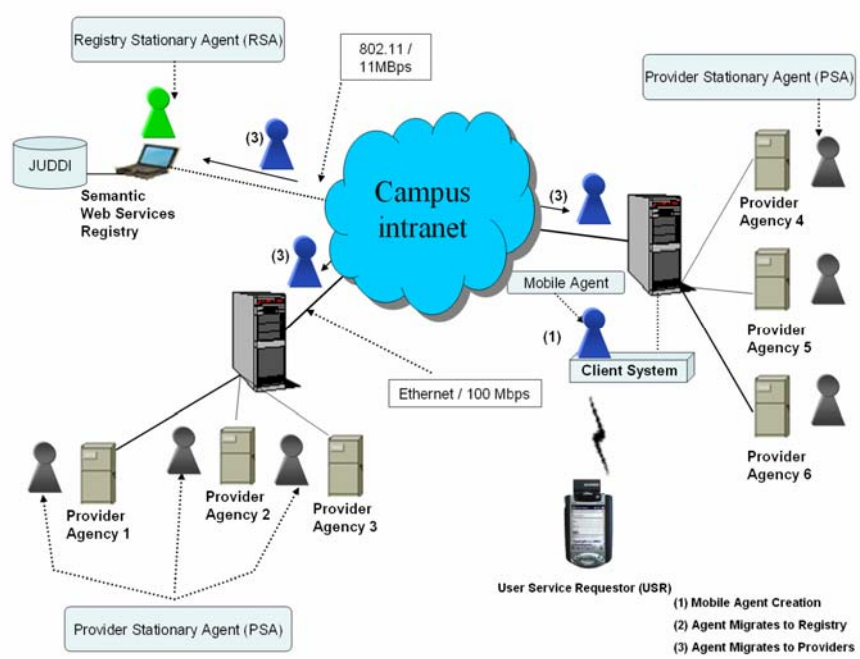

Figure 9: Performance evaluation network topology

Below, we elaborate on the metrics that we adopted in order to assess the performance of the two systems. In Equation (1), Total Service Time (TST_{MA}) (for the SWS&MA platform) is the sum of Registry Interaction Time (*RIT*), Migration of MA to a Service Provider Time (MSPT) and the Interaction Time with this Service Provider (ITSP):

$$TST_{MA} = RIT + MSPT + ITSP \qquad \textbf{(1)}$$

In the WSBM system, Equation (1) has the form:

$$TST_{\text{WSBM}} = RIT + \left\lfloor \frac{N}{2} \right\rfloor * \overline{ITSP} \qquad \textbf{(2)}$$

where the $\overline{ITSP}$ is defined as:

$$\overline{ITSP} = N^{-1} * \sum_{i=1}^{N} ITSP_i \qquad \textbf{(3)}$$

In (2) $ITSP_i$ is the time between service request submission and service results reception.

In Figure 10, the results of the proposed system performance evaluation and comparison against a system implemented using the Conventional WS Business Model are presented. More specifically, the averaged time needed to execute 3 services for the WSBM, is plotted against the time required to invoke only one SWS in the SWS&MA for each service result size (1 KB, 10 KB, 100KB and 1 MB). We observe that the TST in the SWS&MA system is approximately half the TST in the WSBM system, irrespective of the service results size. It should be noted that the RIT in our system is considerably greater than the WSBM system, and this explains that the TST of the SWS&MA is half and not the one third (or even smaller) of the TST of the WSBM system. The high RIT of the proposed SWS&MA framework is attributed to the specific semantic registry implementation and might be less if other semantic registry is used (e.g., OWLS-MX (OWLS-MX, 2007), TUB OWLSM (OWLSM, 2007).

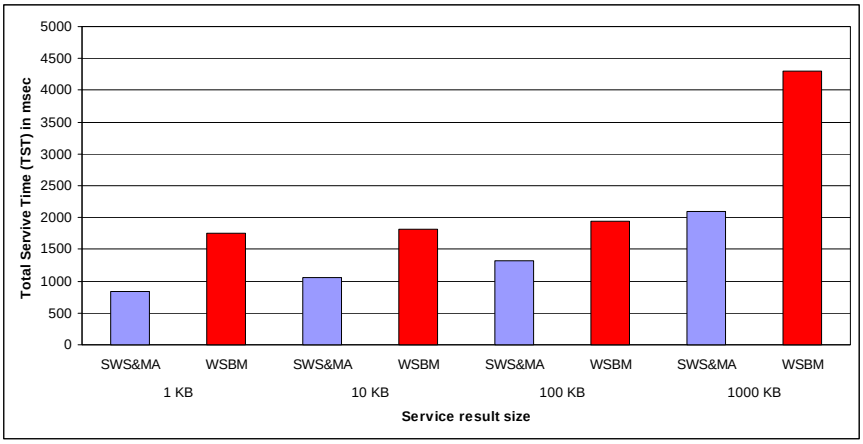

Figure 10: Total Service Time (TST) vs. Service result size

CONCLUSIONS

In this paper we presented a framework that provides wireless access to WS using MA to find and execute WS in the fixed segment. The WS are semantically enriched and are expressed in OWL-S. Furthermore, the proposed system adopts an enhanced WS registry enriched with semantic information that provides semantic matching between service requests submitted and the service description published to them. The advantages of the presented system are: (1) users may invoke a set of services with only one interaction with the fixed network (post the request and receive the results), (2) users do not have to be connected during service discovery and invocation; the results of such operations are downloaded to their mobile devices after their network session re-establishment, (3) service invocations are performed locally or according to the user's specified policies, and unnecessary information is not transmitted over the network leading to better resource utilization, (4) the framework ensures the delivery of the service results to the user, (5) the MA dynamic behaviour improves system robustness and fault tolerance, (6) new services, agents, users and service registries can be easily integrated to the framework, thus, providing an expandable, open system.

Future work includes the study of agent mobility for SWS dynamic invocation and composition that takes network events into account. Network events (e.g., node failures, overloading) occurring while the service invocation is underway, may force the MA to dynamically reschedule its itinerary. The MA will implement routing algorithms that generate itineraries by considering network information published in the WS description, network status and topology.

REFERENCES

ebXML (2007). Retrieved June 1, 2007 from http://www.ebxml.org.

FIPA (2007): Foundation for the Intelligent Physical Agents. Retrieved June 1, 2007 from http://www.fipa.org.

Fuyuki Ishikawa, Nobukazu Yoshioka, Yasuyuki Tahara, and Shinichi Honiden (April 2004) "Mobile Agent System for Web Services Integration in Pervasive Networks", in the proceedings of the International Workshop on Ubiquitous Computing (IWUC 2004), pp.38-47, Porto, Portugal.

Fuyuki Ishikawa, Yasuyuki Tahara, Nobukazu Yoshioka, and Shinichi Honiden, (July 2004b) "Behavior Descriptions of Mobile Agents for Web Services Integration", in the proceedings of the IEEE International Conference on Web Services (ICWS 2004), pp.342--349, San-Diego, CA.

JADE (2007): Java Agent Development Environment. Retrieved June 1, 2007 from http://jade.tilab.com.

jUDDI (2007): Open source Java implementation of the Universal Description, Discovery, and Integration (UDDI) specification for Web Services. Retrieved June 1, 2007 from http://ws.apache.org/juddi/.

Katia Sycara, Massimo Paolucci, Anupriya Ankolekar and Naveen Srinivasan,(July 2004) "Automated discovery, interaction and composition of semantic web services". Journal of Web Semantics, Volume 1.

Lagana Kagal et al (January 2002) "Agents making sense of the semantic web", in the proceedings of the First International Workshop on Radical Agent Concepts, (WRAC 2002), McLean, VA, USA.

Lange, D. and Oshima, M. (1998)"Programming and Deploying Java Mobile Agents with Aglets" . Addison-Wesley.

Li, K., Verma, K., Mulye, R., Rabbani, R., Miller, J., & Sheth,(2006) "Designing Semantic Web Processes: The WSDL-S Approach" In J. Cardoso, A. Sheth (Ed.), Semantic Web Services, Processes and Applications, Springer-Verlag.

Marco Brambilla, Stefano Ceri, Mario Passamani, Alberto Riccio (2004) "Managing Asynchronous Web Services Interactions", in Proceedings of the IEEE International Conference on Web Services (ICWS '04).

Massimo Paolucci, T. Kawamura, T. Payne and K Sycara (June 2002) "Semantic matching of Web services Capabilities", in the proceedings of the International Semantic Web Conference (ISWC2002), Sardinia, Italy.

McIlraith, S., & Martin, D. (2003) "Bringing semantics to web services", IEEE Intelligent Systems, 18(1), 90–93.

Michael Wooldridge (2002) "An Introduction to Multiagent Systems" John Wiley & Sons.

Naveen Srinivasan, Maccimo Paolucci, and Katia Sycara (2004) "Adding OWL-S to UDDI, implementation and throughput", in the proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), San Diego, California, USA.

Nicholas Gibbins, Stephen Harris, Nigel Shadbolt, (2004) "Agent based Semantic Web services" Journal of Web Semantics, Volume 1.

OWL-S (2007): OWL Web Ontology Language for Services (OWL-S). Retrieved June 1, 2007 from http://www.w3.org/Submission/2004/07/.

OWL-S/UDDI Matchmaker Web Interface (2007): Retrieved June 1, 2007 http://www.daml.ri.cmu.edu/ matchmaker/.

OWLSM. (2007). The TUB OWL-S Matcher. Retrieved June 1, 2007 from http://kbs.cs.tu-berlin.de/ivs/Projekte/owlsmatcher/index.html

OWLS-MX. (2007). Hybrid OWL-S Web Service Matchmaker. Retrieved June 1, 2007 from http://www.dfki.de/~klusch/owls-mx/.

Buhler P, et al,(June 2003) “Adaptive Workflow = Web services + Agents”, in the proceedings of the International Conference on Web Services 2003 (ICWS03), Las Vegas, USA.

Buhler P, Jose M. Vidal (May 2004) “Enacting BPEL4WS specified workflows with multi-agent systems”, in the proceedings of the Workshop on Web Services and Agent-Based Engineering (WSABE04), New York, USA.

Pour G, Laad N (July 2006), “Enhancing the Horizons of Mobile Computing with Mobile Agent Components.” Proceedings of the 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering, Software Architecture and Reuse (ICIS-COMSAR’06), pages 225- 230.

R. Montanari, G. Tonti, C. Stefanelli (October 2003) “A Policy-based Mobile Agent Infrastructure”, in the proceedings of the 3rd IEEE International Symposium on Applications and the Internet Workshops (SAINT03) IEEE Computer Society Press, Orlando, USA.

R. Montanari, G. Tonti, C. Stefanelli (October 2003) “Policy-based Separation of Concerns for Dynamic Code Mobility Management”, in the proceedings of the 27th International Computer Software and Applications Conference, (COMPSAC'03), IEEE Computer Society Press, Dallas.

RACER(2007): DL Reasoner. Retrieved June 1, 2007 from http://www.racer-systems.com.

Raghavan, V.V. and Wong, S.K.M (1986). “A Critical Analysis of Vector Space Model for Information Retrieval”, JASIS 37(5), 279-287.

Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., & Fensel, D. (2005). “Web Service Modeling Ontology, Applied Ontology”, 1(1), 77–106.

Sheng-Tzong Cheng et al (February 2002) “A new framework for Mobile Web Services”, in the proceedings of the Symposium on Applications and the Internet (SAINT‘02w) Nara City, Japan.

SWSL Committee (2007): Semantic Web Services Framework (SWSF). Retrieved June 1, 2007 from http://www.daml.org/services/swsf.

Tim Berners-Lee, James Hendler, and Ora Lassila,(2001) “The Semantic Web” Scientific American 2001.

Tomas R. Gruber (June 1993)“A Translation Approach to Portable Ontology Specification”, Knowledge Acquisition, Vol. 5.

Tsetsos V., Anagnostopoulos C., and Hadjiefthymiades S., (2007) "Semantic Web Service Discovery: Methods, Algorithms and Tools", chapter in "Semantic Web Services: Theory, Tools and Applications" (Ed. Dr. Jorge Cardoso), IDEA Group Publishing.

Wassam Zahreddine, Qusay H. Mahmoud (March 2005) “An agent-based approach to composite mobile web services” in the proceedings of the 19th IEEE International Conference on Advanced Information Networking and Applications (AINA05), Taipei, Taiwan.

Ying Huang, Jen-Yao Chung (2003), “A Web Services-based Framework for Business Integration Solutions”, Electronic Commerce Research and Applications, 2(1):15-26. Volume 2.